PARAMETERS, INDEFINITE LOOPS, AND LOOP PATTERNS

CSSE 120—Rose Hulman Institute of Technology

Speed Reading HW Questions?

- It is in HW 10 folder on ANGEL
- You should have already started working on it
- Ask questions if you got stuck
- Finish before Session 12

Review: Python parameter passing

- Formal parameters only receive the **values** of the actual parameters
- Assigning a new value to a formal parameter does not affect the actual parameter
- Python passes actual parameters *by value*
- Can functions in Python mutate parameters?

Functions mutating parameters

- Can we write a function that exchanges the values of its two parameters?

```
# attempt to exchange two integers
def swapInts(x, y):
  x,y = y,x

# attempt to exchange two elements of a list
def swapListElements(list, i, j):
    list[i], list[j] = list[j], list[i]

x,y = 2, 5
print 'Before "swapInts": x=%d, y=%d' %(x, y)
swapInts (x, y)
print 'After "swapInts": x=%d, y=%d' %(x, y)

aList = [3, 4, 5, 6]
print "State of list before swapListElements:",aList
swapListElements(aList, 1, 3)
print "State of list after swapListItems:",aList
```

Modifying Parameters

- How do functions send information back?
  - Return statements
  - *Mutating* parameters
    - Value of actual parameter must be a mutable object
    - State of the mutable object is changed
    - The actual parameter itself is NOT changed since it refers to the same object
    - Parameter is still passed by value

Recap: Two main types of loops

- Definite Loop
  - We know at the beginning of the loop how many times its body will execute
  - Implemented in Python as a **for** loop.
- Indefinite loop
  - The body executes as long as some condition is true.
  - Implemented in Python as a **while** loop.
  - Can be an infinite loop if the condition never becomes False.
- Python's `for line in file:` construct
  - indefinite loop that looks syntactically like a definite loop!

Some indefinite loop patterns

- Interactive loops
- Sentinel loops
- File loops
- post-test loops
- "loop and a half"

Interactive: Make the user count

```
# average1.py
#    A program to average a set of numbers
#    Illustrates counted loop with accumulator

from win_in import *

def main():
    n = win_input("How many numbers do you have? ")
    sum = 0.0
    for i in range(n):
        x = win_input("Enter a number >> ")
        sum = sum + x
    print "\nThe average of the numbers is", sum / n

main()
```

- Not the best idea! (Why not?)

Interactive: Ask user if there is more

```
# average2.py
#    A program to average a set of numbers
#    Illustrates interactive loop with two accumulators
from win_in import *

def main():
    moredata = "yes"
    sum = 0.0
    count = 0
    while moredata[0] == 'y':
        x = win_input("Enter a number >> ")
        sum = sum + x
        count = count + 1
        moredata = win_raw_input("Do you have more numbers (yes or no)? ")
    print "\nThe average of the numbers is", sum / count
```

This initial value makes the loop execute the first time through

- User no longer has to count, but still has a big burden.

Sentinel loop

- User signals end of data by a special "sentinel"value.

```
# average3.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using negative input as sentinel
from win_in import *

def main():
    sum = 0.0
    count = 0
    x = win_input("Enter a number (negative to quit) >> ")
    while x >= 0:
        sum = sum + x
        count = count + 1
        x = win_input("Enter a number (negative to quit) >> ")
    print "\nThe average of the numbers is", sum / count
```

For this program, a negative input number is the sentinel that signals "no more data"

- Note that the sentinel value is not used in calculations.

Non-numeric Sentinel

- What if negative numbers are legitimate values?

```
# average4.py
#    A program to average a set of numbers
#    Illustrates sentinel loop using the empty string as sentinel
from win_in import *

def main():
    sum = 0.0
    count = 0
    xStr = win_raw_input("Enter a number (<Enter> to quit) >> ")
    while xStr != "":
        x = eval(xStr)
        sum = sum + x
        count = count + 1
        xStr = win_raw_input("Enter a number (<Enter> to quit) >> ")
    print "\nThe average of the numbers is", sum / count
```

- **Again note:** sentinel value is not used in calculations.

Interactive loop with graphics

- Display a window that contains a circle and a message saying "Click inside Circle".
- Whenever the user clicks outside the circle, display "You missed!"
- If the user clicks inside the circle, display "Bull's eye!". Then pause and close the window.

File loop

- Use a **for** loop as we have seen before:

```
# average5.py
#     Computes the average of numbers listed in a file.
from win_in import *

def main():
    fileName = win_raw_input("What file are the numbers in? ")
    infile = open(fileName,'r')
    sum = 0.0
    count = 0
    for line in infile:
        sum = sum + eval(line)
        count = count + 1
    print "\nThe average of the numbers is", sum / count

if __name__ == '__main__':
    main()
```

- Also note the conditional execution of **main()**

Individual Exercise on Using loops

- Define function **listAndMax()** in module **listMax.py** that
  - Prompts the user to enter numbers, one at a time
  - Uses a blank line (<ENTER>) as sentinel to terminate input
  - Accumulates the numbers in a list
  - Returns two values:
    - the list of numbers entered in the order they were entered
    - the maximum value of the numbers entered
- Define function **main()** in module **listMax.py** that
  - Calls listAndMax()
  - Prints the list of numbers entered
  - Prints the maximum value of the list of numbers